Assignment 08

Andreas Bäuerle, Tobias Nusser

July 2021

Exercise 2

An important factor for HashJoins is the buffer space which is available and e.g. already existing indexes.

Exercise 3

Bypass selection $\rightarrow$ early tuple elimination to bypass duplicates

a)

$(O_1 \vee O_2) \wedge O_3$

$$C_3 + S_3 \cdot (C_2 + (1 - S_2) \cdot C_1) = 10 + 0.8 \cdot (20 + (1 - 0.2) \cdot 100) = 90$$

b)

$(O_1 \wedge O_2) \vee O_3 = (O_1 \vee O_3) \wedge (O_2 \vee O_3)$
$C_3 + (1 - S_3) \cdot C_2 + S_3 \cdot (C_2 + (1 - S_2) \cdot C_1) = 10 + (1 - 0.8) \cdot 20 + 0.8 \cdot (20 + (1 - 0.2) \cdot 100) = 94$

Exercise 4

The attributes are listed in descending ranking. The smaller the ranking the better suited.

Rank 1 Sort Merge Join

- The sorting stage groups tuples with the same join attribute and lets tuples with similar join attribute values be nearby. Sort orderings allow to find ranges by only searching for the minimum and maximum using binary search each. It also allows for the quick identification of non-equal elements (Search equal, do sequential search before and after).

Rank 2 Index-Nested Loops

- It is basically a draw between Index Nested Loops and Block nested Loops. When using B+-trees as an Index, we can perform all Joins with all predicates. (Index on correct attribute → pretty efficient)
- However, if using a hash index, it can't (see Place 4) answer most of the non-equi joins.
- Since I first thought about B+ Tree Index Joins, I would place Index-Nested Loops on place 2.

Rank 3 Block-Nested Loops

- Is capable to perform the joins. However it doesnt really yield a remarkable performance increase.

Rank 4 Hash Join

- Doesn't support Non-EquiJoins. It cant efficiently compute the cross product as only the hashes of input values are compared.